Electronic Version
Stylesheet Version v1.1.1

Description

Isolated Ordered Regions (IOR) Prefetching and Page Replacement

BACKGROUND OF INVENTION

FIELD OF INVENTION

[0001] The present invention relates generally to prefetching and page replacement. More specifically, the present invention is related to a novel method for prefetching and page replacement based upon grouping nodes of hierarchical data into a plurality of regions.

DISCUSSION OF PRIOR ART

[0002] In transactional systems, finite amounts of pages (free pages) are allocated from memory to hold data from disks. When all free pages are used, that is when all pages contain valid data; when more pages are needed, then certain pages are chosen for replacement. The module that determines which pages to be replaced is the bufferpool component. The bufferpool component tries to keep frequently used pages in memory while replacing others by using well-known protocols. For example, it can try to replace the pages lease recently used in the system (LRU) or the pages most recently used in the system (MRU).

[0003] Bufferpool managers currently use a number of techniques in deciding which pages to replace in the system. One way, as shown in Figure 1, is to use a clock pointer, which is an atomic variable that indexes into a circular array of page descriptors. Within the page descriptors is information that describes bufferpool pages, such as the pointer to the page, a fix count or reference count, description of the contents of the page, and a weight count. The weight count is assigned to a page when the system unfixes (unreferences) a page, and it reflects the probability of that page being referenced in the future. When the free page list is exhausted, the clock pointer is "moved" to point to the pages in the list in a clockwise fashion. While the clock pointer moves from one page descriptor to the next, it decrements the weight count in the current page descriptor. If the clock pointer encounters a page descriptor whose fixed count is zero and whose weight count is below a threshold value, it chooses that page for page replacement. Therefore, the higher weight count assigned to a page, the more likely the page will stay in memory in the bufferpool. Typically, the system assigns high weight counts to pages it knows will be referenced often, such as the root page of an index or a control page.

[0004] Another technique, as shown in Figure 2, uses a linked list of page descriptors pointed to by a head pointer and a tail pointer. Pages are added to the linked list when they're unfixed. If the page is unlikely to be needed in the future, it is added to the head of the list. If it is likely to be referenced in the future, it is placed on the tail of the list. The next page to be replaced will be picked from the head of the list.

[0005] The following references provide for a general teaching with regard to page replacement, but they fail to provide for the claimed invention's robust method and system of page replacement in ordered nodes wherein the nodes are ordered into isolated regions.

[0006] U.S. patent application publication 2003/0018876 A1 provides for virtual memory mapping using region-based page tables, wherein a region register file provides a region identifier for a virtual address in a virtual memory space. The virtual address includes a virtual region number and a virtual page number. A virtual page table lookup circuit is coupled to the region register file to generate a page table entry (PTE) virtual address from virtual address parameters. The virtual address parameters include the virtual address.

[0007] U.S. patent 6,496,912 discloses a system, method, and software for memory management with intelligent trimming of pages of working sets. The computer system has memory space allocatable in chunks, known as pages, to specific application programs or processes. Also disclosed is a trimming method that estimates numbers of trimmable pages for working sets based upon a measure of how much time has elapsed since the memory pages were last accessed by the corresponding application program.

[0008] U.S. patent 6,473,840 discloses a data processing system having a network and method for managing memory by storing discardable pages in a local paging device. A discardable page that is to be removed from the memory is identified. A determination is made as to whether performance will increase by storing the discardable page in a paging device located within the data processing system. If it is determined that performance will increase, the discardable page is marked as a paged discardable page and stored in the paging device locally, wherein this page may be retrieved from the paging device. The paging device may take the form of a paging file, such as a swap file. If space is unavailable within the paging device, the discard-able page may be discarded. These processes may be implemented in a network computer.

[0009] U.S. patent 6,408,368 discloses an operating system page placement to maximize cache data reuse. The operating system designates one or more pages containing critical data, text, or other digital information as hot pages within a physical system memory in the computer system and prevents replacement during execution of various application programs of these hot pages when cached. The operating system inhibits allocation of the conflict pages that would map to cache locations occupied by a cached hot page, thereby preserving the hot page within the cache memory. The conflict pages are placed at the bottom of a free list created in the system memory by the operating system. The operating system scans the free list using a pointer while allocating free system memory space at run-time. The system memory pages are allocated from the free list until the pointer reaches a conflict page. This allows the operating system to prevent the conflict pages from getting cached to the hot page location within the cache memory.

[0010] U.S. patent 6,408,364 describes an apparatus and method for implementing a least recently used (LRU) cache replacement algorithm with a set of N pointer registers that point to respective ways of an N-way set of memory blocks. One of the pointer registers is an LRU pointer, pointing to a least recently used way and another of the pointer registers is a most recently used (MRU) pointer, pointing to a most recently used way. For a cache fill operation in which a new memory block is written to one of the N ways, the new memory block is written into the way (way_n) pointed to by the LRU pointer. All the pointers except the MRU pointer are promoted to point to a way pointed to by respective newer neighboring pointers, the newer neighboring pointers being neighbors toward the MRU pointer.

[0011] U.S. patent 6,347,364 discloses schedulable dynamic memory pinning. An application submits a request for pinning its memory for a certain duration. As compensation, the application may offer other currently mapped pages for replacement. The request may also include the number of pages and the duration of time. The request is granted with the constraint policies which the application is to follow. Such constraint policies include number of pages and length of time the pages may remain pinned in memory. When compensation pages are offered, those pages are replaced in place of the pages which are granted the privilege of being pinned.

[0012] U.S. patent 5,897,660 discloses a method for managing free physical pages that reduces trashing to improve system performance, wherein the claimed invention overcomes the drawbacks of conventional operating system implementations of virtual to physical memory address mapping by providing a method for free physical page management and translation of virtual addresses to physical addresses that increase the effectiveness of the cache memory by reducing the thrashing caused by unfavorable mapping of virtual to physical addresses.

[0013] U.S. patent 5,809,563 discloses a method and apparatus for translating a virtual address into a physical address in a multiple region virtual memory environment. A translation lookaside buffer (TLB) is configured to provide page table entries to build a physical address. The TLB is supplemented with a virtual hash page table (VHPT) to provide TLB entries in the occurrence of TLB misses.

[0014] Transaction systems prefetch pages in anticipation of these pages being referenced in the future. Current systems would prefetch pages that are in some multiple of pages adjacent to the current one being processed. Other mechanisms include using an index which forms an ordered list of page references, which can be used to prefetch pages to be examined. The following references provide for a general teaching with regard to prefetching, but they fail to provide for the claimed invention's robust method and system of prefetching in ordered nodes wherein the nodes are ordered into isolated regions.

[0015] U.S. patent application publication 2002/0103778 discloses a method and system for adaptive prefetching. A cache server may prefetch one or more web pages from an origin server prior to those web pages being requested by a user. The cache server determines which web pages to prefetch based on a graph associated with a prefetch module associated with the cache server. The graph represents all or a portion of the web pages at the origin server using one or more nodes and one or more links connecting the nodes. Each link has an associated transaction weight and user weight.

[0016] U.S. patent application publication 2002/0078165 discloses a system and method for prefetching portions of a web page based on learned preferences. A system and a method for prefetching portions of a web page is based on preferences learned from previous visits to the web page. The disclosed prefetching technique determines whether a user prefers certain sub-pages of the web page and, if so, prefetches these preferred sub-pages prior to the other sub-pages of the web page. The set of preferred sub-pages is generated by analyzing the user's actions during previous visits to the web page.

[0017] U.S. patent 6,385,641 discloses an adaptive prefetching method for use in a computer network and web browsing. The prefetching scheme consists of two modules: a prediction module and a threshold module. After a user's request for a new file is satisfied, the prediction module immediately updates a database of history information, if needed, and computes the access probability for each candidate file where the access probability of a file is an estimate of the probability with which that file will be requested by the user in the near future. Next, the threshold module determines the prefetch threshold for each related server which contains at least one candidate file with nonzero access probability. The threshold is determined in real time based on then current network conditions. Finally, each file whose access probability exceeds or equals its server's prefetch threshold is prefetched. When prefetching a file, the file is actually downloaded if and only if no up-to-date version of the file is available on the local computer; otherwise, no action is taken.

[0018] U.S. patent 6,085,193 discloses a method and system for dynamically prefetching information via a server hierarchy. The method for prefetching data identifies data access patterns and prefetches select information based on dynamic interpretation of the data access patterns. The content server or proxy server identifies data access reference patterns of clients associated with the content server or the proxy server hierarchy. The decision to prefetch select information for the clients is made based on prefetch hint information and prefetch hint values.

[0019] U.S. patent 6,081,799 discloses a method for executing complex SQL queries using index screening for conjunct or disjunct index operations. A query is executed to access data stored on a data storage device connected to a computer. In particular, while accessing one or more indexes to retrieve row identifiers, index matching predicates in the query are applied to select row identifiers and index screening predicates in the query are applied to eliminate one or more selected row identifiers.

[0020] U.S. patent 6,067,565 discloses a technique for prefetching a web page of potential future interest in lieu of continuing a current information download. Described within is a technique that, through continual computation, harnesses available computer resources during periods of low processing activity and low network activity, such as idle time, for prefetching, e.g., web pages or pre-selected portions thereof, into the local cache of a client computer.

[0021] U.S. patent 6,026,474 discloses shared client-side web caching using globally addressable memory. A shared client-side Web cache is provided by implementing a file system shared between nodes. Each browser application stores cached data in files stored in a globally addressable data store. Since the file system is shared, the client-side Web caches are also shared.

[0022] Whatever the precise merits, features, and advantages of the above-cited references, none of them achieves or fulfills the purposes of the present invention.

SUMMARY OF INVENTION

[0023] The present invention provides a system and method for prefetching and replacing pages in storage, wherein the storage retains a plurality of pages and each of the pages comprising a plurality of nodes grouped into one or more regions. The system and method are accomplished via a bufferpool, a prefetcher, and a page replacer. The buffer-pool stores a variable set of pages in memory. The prefetcher recognizes access patterns and usage and fetches pages among the plurality of pages that fit the access patterns and usage. Further, the page replacer works in conjunction with the bufferpool and, during a traversal, weights the variable set of pages to identify a subset to be retained and a remainder to be replaced, wherein the subset includes pages having a high probability of being revisited. The remainder is replaced with a page corresponding to the traversal. Weighting is based upon at least the following numerical values associated with each page in said variable set of pages: number of children, number of parents, and region statistics. The region statistics are any of, or a combination of, the following: minimum step, minimum level, maximum step, or maximum level.

[0024] In an extended embodiment, weighting is additionally based upon identifying pivot pages that define traversals that are not strictly parent-to-child or child-to-parent.

[0025] In another embodiment, the plurality of nodes are associated with a mark-up language based document, such as XML.

[0026] In yet another embodiment, the system and the method are implemented across networks, wherein the across network element is any of the following: local area network (LAN), wide area network (WAN), the Internet, cellular network, or wireless network.

Brief Description of Drawings

[0027] Figures 1-2 illustrate prior art buffer pool managers.

[0028] Figure 3 illustrates an overview of the system of the present invention.

[0029] Figure 4 illustrates how, in a specific example, nodes from an XML document are extracted.

[0030] Figures 5a-c illustrate differing sets of regions formed from a representative XML document.

[0031] Figures 6a and 6b illustrate regions as defined in Figures 5a and 5b, ordered and stored in pages.

[0032] Figures 7a and 7b illustrate a specific example depicting the mapping of nodes in a hierarchically structured document, based upon steps and levels.

[0033] Figure 8 illustrates mapped nodes that are grouped in a plurality of regions, i.e., R1, R2, R3, R4, R5, R6, and R7.

[0034] Figure 9 illustrates the relationships among the calculated step ranges and the various regions of Figure 8.

[0035] Figure 10 illustrates an example that represents a variation of the example illustrated in Figure 8, wherein the variation represents added nodes (and, therefore, added regions).

[0036] Figure 11 illustrates an example that represents another variation of the example illustrated in Figure 8, wherein the variation represents different ordering of regions using the same set of nodes.

[0037] Figure 12 illustrates changes in parameters of regions due to changes in node structure.

[0038] Figure 13 illustrates the effect of the addition of nodes.

[0039] Figure 14 illustrates the effect of nesting levels on various mapped regions.

[0040] Figure 15 illustrates reordering of regions of the document.

[0041] Figures 16a-b illustrate original and region-relative coordinates associated with nodes in various regions.

[0042] Figure 17 illustrates a scenario wherein post order traversal (POT#) numbers can be computed for each of the nodes during traversal to identify containment relationships among nodes.

[0043] Figures 18a-b illustrate a set of nodes and their respective mapping based on the method of assigning a step number every time a child node is descended.

[0044] Figure 18c illustrate regions R1 through R6 formed from the set of nodes.

[0045] Figures 18d-f illustrate the effect of modifications on the above-mentioned parameters with respect to regions R1 through R6.

[0046] Figure 19 illustrates nodes grouped by node descendant regions.

[0047] Figure 20 illustrates the present invention's method for prefetching pages based upon access patterns.

[0048] Figure 21 illustrates the present invention's method for page replacement in bufferpools.

[0049] Figure 22 illustrates the present invention's method for unfixing a pivot page.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0050] While this invention is illustrated and described in a preferred embodiment, the invention may be produced in many different configurations. There is depicted in the drawings, and will herein be described in detail, a preferred embodiment of the invention, with the understanding that the present disclosure is to be considered an exemplification of the principles of the invention and the associated functional specifications for its construction and is not intended to limit the invention to the embodiment illustrated. Those skilled in the art will envision many other possible variations within the scope of the present invention.

[0051] The present invention presents a robust system and method for prefetching and replacing pages in an ordered node structure, wherein the node structure is representative of hierarchical data associated with a document and the ordering is based on the concept of isolated ordered regions. The coordinates of nodes are maintained by associating each node with coordinates relative to a containing region. Modifications to nodes within a region only affect the nodes in that region -- not nodes in other regions. Traversals that retrieve information from the nodes can rebase the coordinates from their containing region and return results with a total order.

[0052] Figure 3 illustrates an overview of the node ordering system *300* used in conjunction with the present invention. Document *302*, containing hierarchical data (e.g., a document in a mark-up language-based format such as XML) is stored by creating nodes, via node generator *304*, that parse and represent the entities of the document and the relationships that exist among the entities. The nodes are grouped into regions (i.e., Region R_1 through R_n) via region grouper *306* and are formatted, via formatter *308*, and written out to pages. As the nodes are created, they are grouped into regions in various ways depending upon anticipated access patterns and usage. Each region is formatted and stored into pages managed by the system. The present invention's prefetcher *310* recognizes access patterns and usage and fetches pages that fit the recognized patterns. The present invention's page replacer *312* works in conjunction with a bufferpool and identifies the probability of revisiting traversed nodes, wherein the page replacer replaces pages (in bufferpool) with least probability of being revisited. There can be more than one region written to a page. Figure 4 illustrates how, in a specific example, nodes from an XML document *400* are extracted.

[0053] It should be noted that although throughout the figures and specification an XML document has been used to illustrate various examples, other hierarchically-ordered documents, such as documents in mark-up language formats such HTML, can be equally used in conjunction with the present invention. Therefore, the scope of the present invention should not be limited by the type of hierarchically-ordered document.

[0054] Figures 5a-c illustrate differing sets of regions formed from a representative XML document *400*. In this example, a set of nodes representative of an XML document can be broken, in Figure 5a, into three regions *502*, *504*, and *506*. Similarly, in Figures 5b and 5c, the same set of nodes is broken into regions *508-514* and *516-524*,respectively. Figures 6a and 6b illustrate the next stage, wherein the regions as defined in Figures 5a and 5b are ordered and stored in pages.

[0055] Each region has a number of values associated with it wherein these values are computed using algorithms, to be outlined later. Each region has a Minimum Step (Min Step), Minimum Level (Min Level), Maximum Step (Max Step), Maximum Level (Max Level), and Step Range. The Min Step, Min Level, Max Step, and Max Level numbers give a region its dimensions and order within the document.

[0056] Figures 7a and 7b illustrate a specific example depicting the mapping of nodes in a hierarchically-structured document based upon steps and levels. The following algorithm, based on a set of rules, is used in such a mapping:

[0057]

```
For every descendant
    $Level = Level + 1$
For every ancestor
    $Level = Level - 1$
For every [>1st] child node
    $Step=Step+1$
```

[0058] Next, as in Figure 8, the mapped nodes are grouped in a plurality of regions: R1 *800*, R2 *802*, R3 *804*, R4 *806*, R5 *808*, R6 *810*, and R7 *812*. Specific grouping patterns are used for illustrative purposes only, and it should noted that other groupings of the same set of nodes are envisioned (as illustrated in Figures 5a-c) in conjunction with the present invention. Also, as mentioned earlier, specific groupings of regions can depend on anticipated access patterns and usage.

[0059] The regions of Figure 8 are ordered based on ascending Min Step and ascending Min Level. Table 1, provided below, depicts a list of regions ordered based upon ascending Min Step and Min Levels.

[0060]

	Nesting Level	Min Step	Min Level	Max Step	Max Level
R1	0	0	0	2	2
R2	0	1	3	3	4
R3	0	2	5	3	6
R4	0	4	1	5	3
R5	0	6	2	9	2
R6	0	6	3	8	4
R7	0	10	2	12	2

Table 1

[0061] Regions are then grouped into Step Ranges. Regions within a Step Range have Min Step and Max Step values that do not overlap with that of regions in other Step Ranges. The algorithm below illustrates how to compute the Step Ranges in a set of regions:

[0062]

```
LET POT# = Post Order Traversal Number
        LEVEL = Node Level Coordinate
        STEP = Node Step Coordinate
FOR EVERY Node N1 AND Node N2

IF (N1 POT# > N2 POT#) AND
   (N1 LEVEL < N2 LEVEL) AND
   (N1 STEP <= N2 STEP)
THEN
N1 CONTAINS N2
```

[0063] Table 2, provided below, depicts Step Ranges (calculated based upon the above-mentioned algorithm) of regions of Figure 8 and Table 1.

[0064]

Region (Parameters)	Step Ranges
R1[0,0,0,2,2]	0
R2[0,1,3,3,4]	0
R3[0,2,5,3,6]	0
R4[0,4,1,5,3]	4
R5[0,6,2,9,2]	6
R6[0,6,3,8,4]	6
R7[0,10,2,12,2]	10

Table 2

[0065] The relationship among the calculated Step Ranges and the various regions of Figure 8 (i.e., R1 *800*, R2 *802*, R3 *804*, R4 *806*, R5 *808*, R6 8*10*, and R7 *812*) are shown in Figure 9. In Figure 9, SR6 represents regions that have a step range equal to 6, which in this case encompasses regions R5 (*808* of Figure 8) and R6 (*810* of Figure 8).

[0066] Figure 10 illustrates an example that represents a variation of the example illustrated in Figure 8, wherein the variation represents added nodes (and therefore, added regions). In addition to the regions of Figure 8, Figure 10 further includes regions R8 *1002*, R9 *1004*, R10 *1006*, and R11 *1008*. The addition of these regions (R8-R11) changes the Step Ranges associated with all regions depicted in Figure 10. Table 3, provided below, depicts Step Ranges (calculated based upon the above-mentioned algorithm) of regions of Figure 10.

[0067]

Region (Parameters)	Step Ranges
R1[0,0,0,2,2]	0
R2[0,1,3,2,4]	0
R8[0,2,5,3,6]	0
R3[0,2,7,3,8]	0
R9[0,3,3,4,5]	0
R11[0,5,4,7,5]	5
R10[0,8,3,10,5]	8
R4[0,11,1,12,3]	11
R5[0,13,2,16,2]	13
R6[0,13,3,15,4]	13
R7[0,17,2,19,2]	17

Table 3

[0068] Figure 11 illustrates an example that represents another variation of the example illustrated in Figure 8, wherein the variation represents different ordering of regions using the same set of nodes. The new set of regions of Figure 11 includes: R1 *1100*, R2 *1102*, R3 *1104*, R4 *1106*, R5 *1108*, R6 *1110*, and R7 *1112*. The regions of Figure 11 are ordered based upon ascending Min Step and ascending Min Level. Table 4, provided below, depicts a list of regions of Figure 11 ordered based upon ascending Min Step and Min Levels.

[0069]

	Nesting Level	Min Step	Min Level	Max Step	Max Level
R1	0	0	0	1	2
R3	0	1	3	4	3
R2	0	2	2	4	2
R4	0	2	4	3	6
R5	0	5	1	7	2
R7	0	6	3	8	4
R6	0	8	2	12	2

Table 4

[0070] Once the Step Ranges are determined, the regions are re-ordered. The previous entries ordered based on Min Step, Min Level are now ordered based on Step Range, Min Level, Min Step. In the above example, regions R2 *902*, R3 *904*, R6 *910*, and R7 *912* are reordered. This change accurately reflects the parent-child relationship of the regions within a Step Range. Table 5, provided below, depicts a list of regions of Figure 11 reordered based upon Step Range, Min Levels, Min Step:

[0071]

	Nesting Level	Min Step	Min Level	Max Step	Max Level
R1	0	0	0	1	2
R3	0	1	3	4	3
R2	0	2	2	4	2
R4	0	2	4	3	6
R5	0	5	1	7	2
R7	0	6	3	8	4
R6	0	8	2	12	2

Table 5

[0072] Figure 12 illustrates changes in parameters of regions due to changes in node structure. When nodes added to or deleted from a region(s) are added to or deleted from a document, the Step Range, Min Level, Min Step, Max Level, and Max Step values of neighboring regions are affected. For regions in the same Step Range, retraversal of the regions needs to be performed to recompute the Min Level, Min Step, Max Level, and Max Step values. For regions in Step Ranges that follow, only the Min Step and Max Step values need adjustment.

[0073] To defer retraversal of regions during modifications, regions affected can be assigned to another nesting level. When this happens, the original set of regions retains its dimensions with respect to other regions in the same nesting level, while a new set of regions with a higher nesting level is created.

[0074] For example, as shown in Figure 13, nodes were added to region R2 of Figures 8 and 9 which modified its dimensions and caused other regions R2 (of nesting level 1) *1300*, R8 *1302*, R9 *1304*, and R10 *1306* to be created. The index used to maintain the order of the regions still preserves the order of R2 *1308* with respect to R3 *1309* in nesting level 0, but now R2 *1310*, R9 *1312*, R10 *1314*, and R8 *1316* of nesting level 1 are ordered in-between (shown as tabbed entries in Figure 13).

[0075] The regions of the new nesting level have Min Level, Min Step, Max Level, and Max Step values that are computed with respect to their parent nesting level region. For example, R8 *1316* in nesting level 1 is two levels and one step away from the axis of R2 *1308* in nesting level 0. It should be noted that, in Figure 13, the axis of R2 *1300* starts off a new Step count (from 0 to 6) and a new Level count (from 0 to 3) in nesting level 1.

[0076] Further modifications to the document cause more regions of higher nesting levels to be created, all based of a parent level region. For example, as illustrated in Figure 14, R9 of nesting level 1 *1400* grows, causing R9 *1402* and R11 *1404* of nesting level 2 to be created. Again, it should be noted that regions in ancestor nesting levels do not change.

[0077] After the retraversal of regions within the Step Range and the adjustment of Min Step and Max Step values of the regions in the Step Ranges that follow, the regions of the document will be reordered, as shown in Figure 15.

[0078] Traversals that extract information from nodes require an order for each of the nodes in the document. Step and Level values can be associated with each of the nodes to impose an order for all nodes. These values are computed relative to the Step and Level values of the containing region. Modifications in the document only affect the region coordinates which are maintained in a Region Index -- not the node coordinates. In the example illustrated in Figure 16a, the coordinate of the H node in Region R2 with Step 2, Level 3 is Step 0, Level 1 relative to the region. Therefore, H nodes" absolute coordinate is Step 2, Level 4. Figure 16b illustrates original and region-relative coordinates associated with nodes in Regions R1-R4.

[0079] Figure 17 illustrates a scenario wherein post-order traversal (POT#) numbers can be computed for each of the nodes during traversal to identify containment relationships among nodes. Given the Step, Level, and POT# values of two nodes, the system can determine whether one node contains the other node. In the example below, the node with Step 5, Level 2, POT# 19 contains the node with Step 7, Level 4, POT# 13; but the node with Step 10, Level 1, POT# 24 does not contain the node with Step 7, Level 4, POT# 13. The algorithm for identifying if a node N1 contains another node N2 is given below:

[0080]

```
LET POT# = Post Order Traversal Number
         LEVEL = Node Level Coordinate
         STEP = Node Step Coordinate
FOR EVERY Node N1 AND Node N2


IF (N1 POT# > N2 POT#) AND
   (N1 LEVEL < N2 LEVEL) AND
   (N1 STEP <= N2 STEP)
THEN
N1 CONTAINS N2
```

	Nesting Level	Min Step	Min Level	Max Step	Max Level
R1	0	0	0	1	2
R3	0	1	3	4	3
R2	0	2	2	4	2
R4	0	2	4	3	6
R5	0	5	1	7	2
R7	0	6	3	8	4
R6	0	8	2	12	2

Table 5

[0081] It should be noted that although algorithms identified above (e.g., in the discussions of Figure 7a-7b) are based on a set of rules associated with how steps are computed, other variations of computing steps are also within the scope of the invention. For example, the algorithm described in relation to Figure 7a-b has problems with updates, as it could move nodes from one region out into another region during updates. For example, if the document were a, b, c, and d, where a->b, and b->c and b->d, then a, b, and c would be assigned step 0 and d assigned step 1; but during updates, if c were removed, d would need to be moved up one step, potentially into a different region from where it was originally. A simpler way of "counting" steps is to monotonically increase the step number every time a child node is descended. That is, if the document is a->b->c, b->d, a->e, then the step numbering would be a (1), b (2), c (3), d (4), e (5). This step numbering scheme is the same as that for preorder traversal of a tree of nodes and is illustrated in Figures 16a-f.

[0082] Figures 18a-b illustrate a set of nodes and their respective mapping based on the above-mentioned method of assigning a step number every time a child node is descended. Figure 18c illustrates regions R1 through R6 formed from the set of nodes wherein the parameters of R1 through R6 are provided below in Table 6:

[0083]

	Min Step	Min Level	Max Step	Max Level
R1	0	0	6	3
R2	7	0	12	6
R3	13	0	14	3
R4	15	0	22	2
R5	18	3	21	4
R6	23	0	26	3

Table 6

[0084] Figures 18d-f illustrate the effect of modifications on the above-mentioned parameters with respect to regions R1 through R6. Thus, as can be seen in the examples illustrated in Figures 18a-f, the specific algorithm used to compute the steps can vary and, hence, should not be used to limit the scope of the present invention.

[0085] Similarly, regions can be grouped in a varying fashion. For example, as in Figure 19, nodes can be grouped based upon node descendant regions. Node descendant regions are regions that contain all nodes that are descendents of a particular node. Figure 19 illustrates Node Descendant Regions NDR1-NDR4, whose parameters are summarized in Table 7 below:

[0086]

	Min Step	Min Level	Max Step	Max Level
NDR1	2	2	2	2
NDR2	5	2	12	6
NDR3	17	2	18	3
NDR4	19	2	21	4

Table 7

[0087] Thus, as can be seen in the example above, various rules can be used to identify regions among a set of nodes and, hence, such rules should not be used to limit the scope of the present invention.

[0088] The present invention's robust system and method for prefetching and page replacement takes advantage of the above-mentioned node structure representative of hierarchical data associated with a document, wherein ordering is based on the concept of isolated ordered regions.

[0089] For example, returning to the example depicted in Figures 8 and 9, after the regions of a document have an order, region access patterns that resemble certain orders in the index can be recognized. This is used to prefetch the pages where the next set of regions in the order is stored. For example, in Figure 20, two access patterns are recognized by the system. The Prefetch Depth-wise pattern resembles a region access pattern with "slow" increasing steps, while the Prefetch Breath-wise pattern resemble a region access pattern with "slow" increasing levels. The index can be used to find the next set of regions and, consequently, the pages where the regions are stored, so that the pages can be prefetched.

[0090] In addition to the advantage of a robust prefetch algorithm, the present invention also provides for a page replacement algorithm based upon node order and hierarchical data. Hierarchical data can be stored by placing related sets of nodes into pages. While processing the nodes, a number of statistics can be gathered for each page. These statistics include the number of child pointers and parent pointers as well as the nodes relative order within the hierarchy.

[0091] Using these statistics, the system can tell the bufferpool component how the degree of probability that the pages traversed will be needed again in the future. This will cause the bufferpool component to keep the "most likely to be referenced" pages in memory longer.

[0092] The present invention can be used for the page replacement of bufferpool pages that contain nodes of a markup-based document, such as an XML document. As mentioned earlier, an XML document is stored when the system creates nodes that represent the entities of the document and the nodes are written out to pages. The nodes created are grouped (into regions) in various ways depending on anticipated access patterns. As regions are created, they are written out to bufferpool pages. Also, as mentioned earlier, a number of statistics are kept for each page. The Num Children and Num Parent values reflect the number of edges incident to the page. The Min Step and Min Level values reflect the order of the regions contained in the page with respect to the hierarchy. Regions with high Num Children and Num Parent values tend to be pages that are referenced often.

[0093] A region's Min Step/Min Level value reflects the region's order within the hierarchy. Regions with low Min Step/Min Level values tend to be regions which will be referenced often when the traversal moves from low Step/Level regions to high Step/Level regions. On the other hand, regions with high Min Step/Min Level values tend to be regions which will be referenced often when traversals move from high Step/Level regions to low Step/Level regions.

[0094] Using the Num Children, Num Parent, Min Step, and Min Level values associated with a page, the system can provide more information for the bufferpool component during page unfix. For example, in a traversal from a parent page to a child page where the parent pages Min Step/Min Level value is low and Num Children value is high, the system can unfix the parent page with a high weight count, as it is likely that the page will be referenced again. Subsequent pages will then have lower weights compared to their parent. This is illustrated in Figures 21a-j.

[0095] Figures 21a-j show chronologically, from left to right and top to bottom, how bufferpool pages are used to read in pages on disk and how pages unlikely to be referenced again are chosen for replacement. In areas where there is an asterisk (*), a LRU-based page replacement algorithm would have chosen a bad candidate.

[0096] Figure 21a illustrates an initial condition depicting pages (more specifically, 11 pages) stored on a disk. Figure 21b illustrates a node *2100* that represents a page that is latched (or pinned) in memory. For the purposes of this example, it is assumed that, at any given time, four pages from the disk are present in a bufferpool. For example, Figure 21c illustrates a scenario wherein four pages *2102*, *2104*, *2106*, and *2108* are currently in use and are present in the bufferpool, wherein page *2108* also happens to be the page that is latched in memory.

[0097] Next, in Figure 21d, the page that is latched in memory is changed to page *2110*. Based upon the present invention, the page that gets replaced in the buffer is page *2112*. This decision is made based upon using the number of children nodes, number of parent nodes, min step, and min level values associated with a page. As with the prior art systems that use the least recently used (LRU) algorithm, page *2114* would have been dropped out of the bufferpool, as it was the least recently used page. But, the present invention's method takes into account the fact that page *2114* has a plurality of children and, hence, associates a higher probability that the page *2114* will be accessed again in the future. Thus, page *2114* is kept in the bufferpool, while page *2112* (which does not have any children) is dropped from the bufferpool.

[0098] It should be noted that, although systems that drop pages based upon the most recently used (MRU) algorithm would have also picked page *2112* as the page to dropped from the bufferpool, such systems do not benefit from the decision being made based upon taking into account factors such as number of children nodes, number of parent nodes, min step, and min level values associated with a page. Additionally, as will be seen later, the present invention's method does not mirror the results of neither the MRU not LRU algorithms.

[0099] Figures 21e-g further illustrate how the page latched in memory changes traverses within the hierarchy (i.e., latched page *2116* in Figure 21e, latched page *2118* in Figure 21f, and latched page *2120* in Figure 21g). Figure 21h illustrates a scenario wherein page *2122* is latched in memory. Next, in Figure 21i, the page that is latched in memory is changed to page *2124*. Based upon the present invention, the page that gets replaced in the buffer is page *2126*. This decision is made based upon taking into account: the number of children nodes, number of parent nodes, min step, and min level values associated with a page. A system based upon the LRU scheme would have picked page *2128* as the page to be replaced in the bufferpool. Similarly, when the latched page is page *2130*, the present invention replaces page *2132* in the bufferpool, whereas LRU schemes would have replaced page *2134*. Also, an MRU scheme would have replaced page *2136*.

[0100] In cases where the traversal is not purely parent-to-child or child-to-parent, the plan generator can leave hints about certain pages (called pivot pages) which, when reached, can change the direction of the traversal and will likely be needed again after it is unfixed. These pages usually contain nodes involved in predicate evaluations. Therefore, when it comes time to unfix a pivot page, the system can unfix it with a higher weight. Figures 22a-g illustrate this aspect.

[0101] Figures 22a-g show how a page that contains nodes, and has pointers (*2200*, *2202*, *2204*) to other pages which contain their children nodes, might be prematurely replaced by a transactional system (specifically those pages marked with an asterisk) with a page replacement module that uses an LRU. But, the present invention provides for a system and method that properly assign weights to pages by considering various factors (such as, but not limited to, the number of children, region statistics, or hints left by the compiler) to identify which pages are likely to be kept in memory longer for retraversals to be accomplished efficiently.

[0102] This aspect of the present invention is best shown in Figures 22f-g. In Figure 22f, page *2206* is latched in memory, while pages *2208*, *2210*, and *2202* are in an unlatched (or unpinned) condition. Next, in Figure 22g, page *2212* is latched in memory. Now, instead of removing pointer page *2200* from the bufferpool, the present invention notices that this is a pointer page and retains it in an unlatched state (as the probability of returning to a pointer page is high). Instead, page *2214* is removed from the bufferpool.

[0103] It should be noted that the number of pages depicted in for illustrative purposes only and should not be used to limit the scope of the present invention.

[0104] One aspect of this approach that makes it different from traditional LRU/MRU-based techniques is its bias towards keeping pages in memory as opposed to a bias towards throwing pages out of memory. Because of the nature of hierarchical data, access patterns will tend to visit pages that have non-leaf nodes multiple times -- sometimes after pages with leaf nodes have been visited. As a consequence, pages with non leaf nodes need to stay in memory longer. That is why this approach, oriented toward that kind of access pattern, is more beneficial than LRU/MRU.

[0105] One difference hierarchical data accesses have over index node accesses lies in the fact that non-leaf index nodes tend to not be reaccessed and most of the time result in index leaf nodes accesses, whereas accesses to hierarchical data doesn't necessarily result in leaf page accesses.

[0106] Additionally, the present invention provides for an article of manufacture comprising computer-readable program code contained within implementing one or more modules for ordering nodes in a document (e.g., XML document). Furthermore, the present invention includes a computer program code-based product, which is a storage medium having program code stored therein which can be used to instruct a computer to perform any of the methods associated with the present invention. The computer storage medium includes any of, but is not limited to, the following: CD-ROM, DVD, magnetic tape, optical disc, hard drive, floppy disk, ferroelectric memory, flash memory, ferromagnetic memory, optical storage, charge coupled devices, magnetic or optical cards, smart cards, EEPROM, EPROM, RAM, ROM, DRAM, SRAM, SDRAM, or any other appropriate static or dynamic memory or data storage devices.

[0107] Implemented in computer program code-based products are software modules for: (a) instructing a computer to store a variable set of pages in memory; (b) recognizing access patterns and usage and fetching pages among the plurality of pages that fit the access patterns and usage;

and (c) upon traversals within the plurality of pages: (i) instructing the computer to retain a subset of the variable set to include pages having a high probability of being revisited; and (ii) instructing the computer to dynamically replace remainder of the variable set with a page corresponding to said traversal; and during each traversal, one or more modules weights the variable set of pages to identify the subset to be retained and the remainder to be replaced, wherein the weighting is based upon at least the following numerical values associated with each page in said variable set of pages: number of children, number of parents, minimum step, and minimum level.

CONCLUSION

[0108] A system and method has been shown in the above embodiments for the effective implementation of a system and method for prefetching and page replacement. While various preferred embodiments have been shown and described, it will be understood that there is no intent to limit the invention by such disclosure but, rather, it is intended to cover all modifications falling within the spirit and scope of the invention, as defined in the appended claims. For example, the present invention should not be limited by number of nodes in bufferpool, size of bufferpool, type of hierarchically-ordered document, the type of algorithm used to calculate the step, number of nodes, number of levels, number of steps, number and shape of regions, software/program, or computing environment.

[0109] The above enhancements are implemented in various computing environments. For example, the present invention may be implemented on a conventional IBM PC or equivalent, multi-nodal system (e.g., LAN) or networking system (e.g., Internet, WWW, wireless). All programming and data related thereto are stored in computer memory, static or dynamic, and may be retrieved by the user in any of: conventional computer storage, display (i.e., CRT), and/or hardcopy (i.e., printed) formats. The programming of the present invention may be implemented by one of skill in the art of mark-up-based languages and database programming.